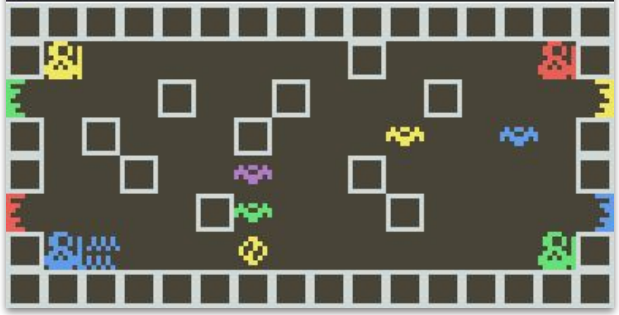
Atelier jeu vidéo

Création d'un jeu vidéo Python pour calculatrices Casio Graph

Exemples de jeux codéveloppés en langage Python pour calculatrices Casio Graph :

SynchroD *(rentrée 2021)*



RPG **Alrys** *(rentrée 2021)*

Doom-like **Pykaster3D** *(rentrée 2022)*

Construisons aujourd'hui ensemble
un tout nouveau jeu…

It’s… Flappy Bird

Jeu mobile tactile sorti en 2013 pour Android et iOS.

Emprunte à l’univers graphique de Super Mario.

Il s’agit de rythmer correctement les battements d’ailes d’un oiseau afin de lui faire franchir un nombre maximal de tuyaux.

Une seule action (le battement d’ailes) ; le jeu sur calculatrice n’aura besoin que d’1 seule touche.

Le jeu vidéo sur calculatrice

Un jeu vidéo sur calculatrice est décomposable en **3** éléments plus ou moins distincts :

- **gestion des entrées** : vérifie l'état des touches du clavier + active les actions correspondantes
- **moteur physique** : représente les données de l'environnement du jeu, les maintient à jour, effectue les calculs destinés à simuler des phénomènes (déplacements, chutes, etc.)
- **moteur graphique** : dessine les différentes formes simples destinées à composer les différents éléments visuels 2D ou 3D du jeu (segments, rectangles, triangles, etc.)

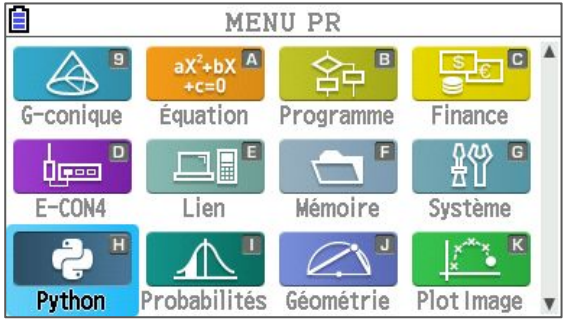
Le code d'un jeu vidéo sur calculatrice se compose usuellement d'une boucle dont le corps fait appel à ces 3 éléments.

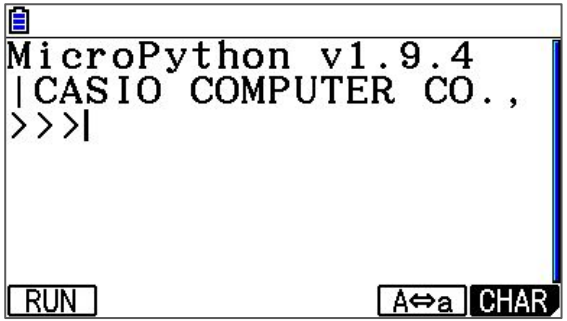
L’application Python des Casio Graph

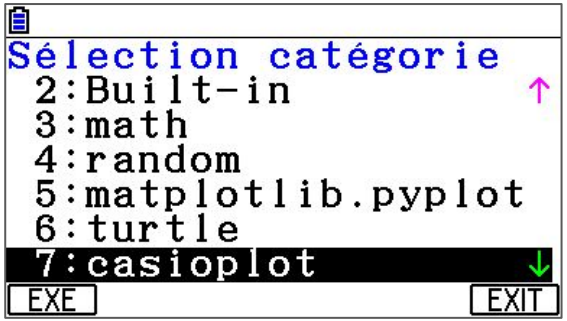
Interpréteur **MicroPython** en version **1.9.4**

5 bibliothèques intégrées :

- **3** bibliothèques standard **Python 3** :
  - **math**
    fonctions mathématiques
  - **random**
    génération de données aléatoires
  - **turtle**
    tracé par déplacements (à la Scratch / Logo)
- **1** bibliothèque tierce populaire :
  - **matplotlib** (limitée à **matplotlib.pyplot**)
    tracé de diagrammes dans des repères
- **1** bibliothèque spécifique aux calculatrices **Casio Graph** :
  - **casioplot**
    tracé par pixels ; utilisé par turtle et matplotlib







Le fonctionnement de la bibliothèque casioplot

Travaille en **double buffering** :

- les tracés demandés sont d'abord écrits dans une zone non affichée de la mémoire, sorte d'image cachée
- puis, sur demande, copie des données vers l'écran pour tout afficher d'un coup

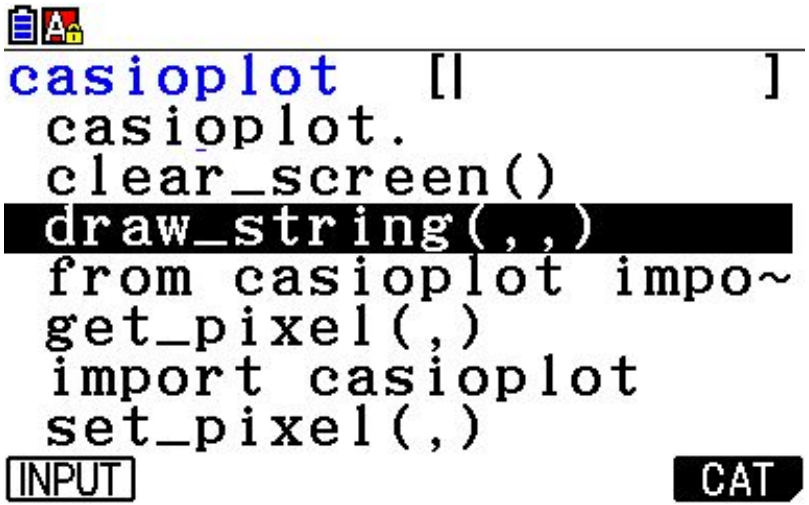
Avantages du double buffering:

- masque les affichages intermédiaires inesthétiques entre deux écrans (tracés progressifs ou partiels, clignotements, etc.)
- fluidité des animations

Les fonctions de la bibliothèque casioplot

5 fonctions intégrées :

- **4** fonctions agissant sur l'image cachée :
  - **clear_screen()**
    Efface l'image en blanc
  - **draw_string(x,y,texte,couleur,police)**
    Écrit des chaînes de caractères sur l'image
  - **get_pixel(x,y)**
    Retourne la couleur d'un pixel de l'image
  - **set_pixel(x,y,couleur)**
    Change la couleur d'un pixel de l'image
- **1** fonction agissant sur l'écran :
  - **show_screen()**
    Affiche l'image préparée à l'écran



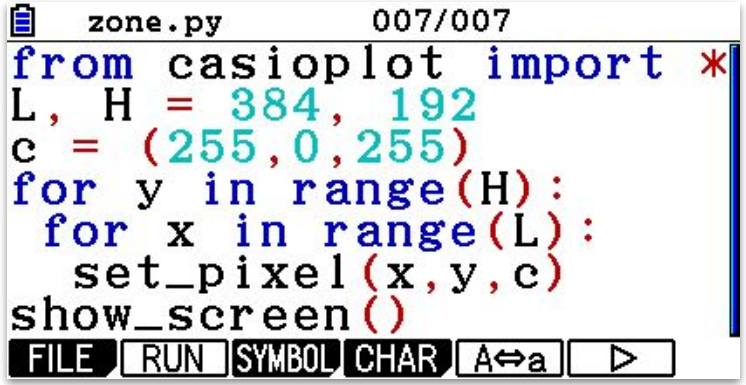
La zone graphique de casioplot

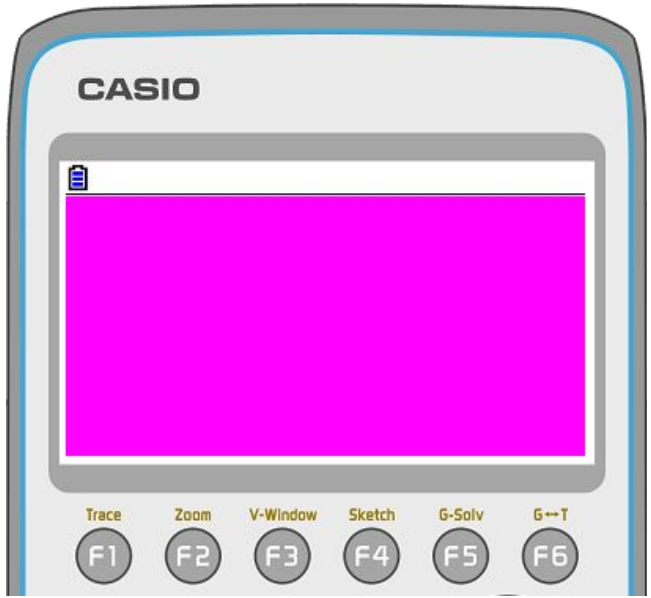
Contrôle une zone graphique de :

- **128*64** pixels sur **Graph 35+E II**
  plein écran
- **384*192** pixels sur **Graph 90+E**
  *écran de 396*224 pixels, limité par barre de statut en haut et bandes sur les autres côtés*

Pixels identifiés par leurs coordonnées comptées à partir du coin supérieur gauche de la zone, de gauche à droite et de haut en bas

Exemples de coordonnées de pixels	**Graph 90+E**	**Graph 35+E II**
supérieur gauche	(0, 0)	(0, 0)
Supérieur droit	(383, 0)	(127, 0)
Inférieur gauche	(0, 191)	(0, 63)
Inférieur droit	(383, 191)	(127, 63)





Format de couleur casioplot

Listes de 3 éléments précisant la composition de la couleur selon 3 couleurs primaires :
(Rouge, Vert, Bleu)

Chaque élément est l'intensité de la couleur primaire ; un entier **8 bits** autorisant 2^8= **256** valeurs différentes : de **0** (absence de la couleur primaire) jusqu'à une intensité maximale de **255**.

La couleur en résultant correspond à la synthèse additive des 3 composantes.

Codage sur 8+8+8= **24 bits**, permettant de décrire 2^{24}= **16777216** couleurs différentes.

exemples de couleurs	composante rouge	composante verte	composante bleue	liste pour casioplot
rouge	255	0	0	(255,0,0)
vert	0	255	0	(0,255,0)
bleu	0	0	255	(0,0,255)
blanc	255	255	255	(255,255,255)
noir	0	0	0	(0,0,0)
jaune	255	255	0	(255,255,0)
orange	255	127	0	(255,127,0)
marron	127	63	0	(127,63,0)
magenta	255	0	255	(255,0,255)
violet	127	0	255	(127,0,255)
cyan	0	255	255	(0,255,255)
gris	127	127	127	(127,127,127)

Les couleurs sur Graph 90+E

L'écran **Graph 90+E** affiche les couleurs les plus proches sur l'échelle **RVB565**, codant :

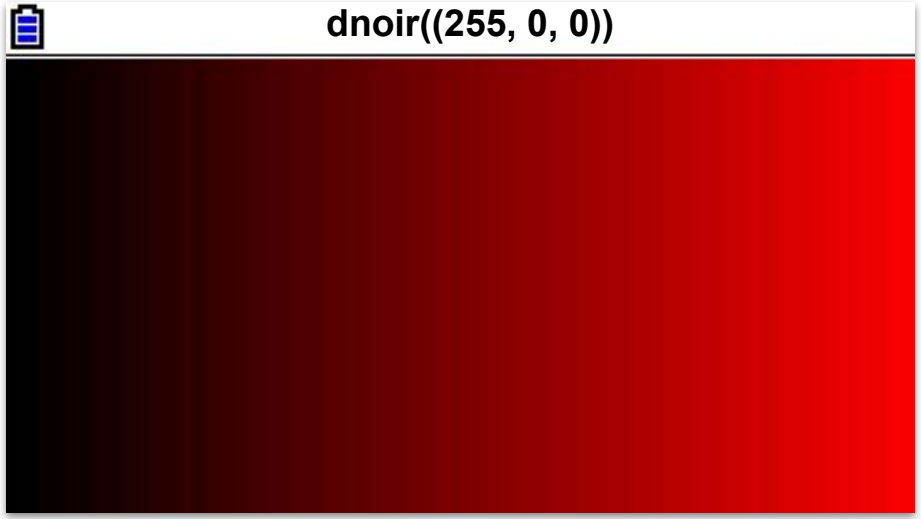
- la composante rouge sur 5 bits (2^5= 32 teintes différentes)
- la composante vert sur 6 bits (2^6= 64 teintes différentes)
- la composante bleu sur 5 bits (2^5= 32 teintes différentes)

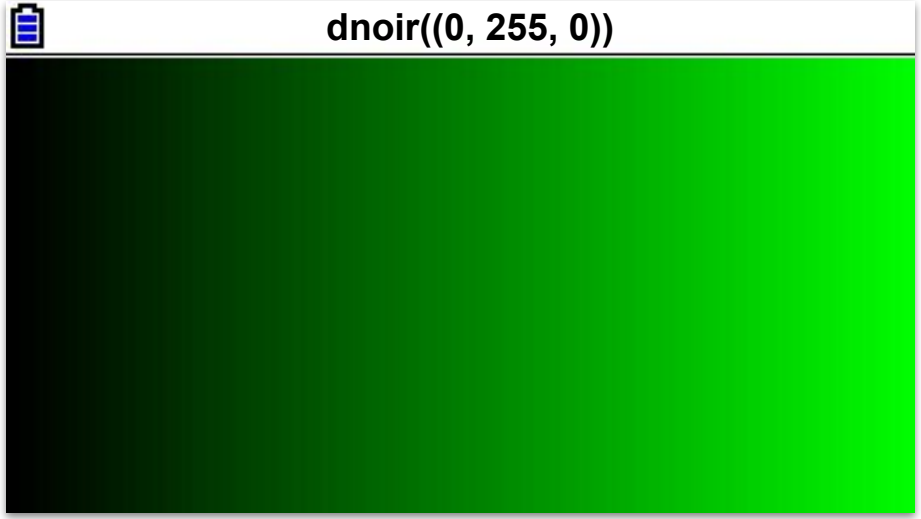
Par combinaison affichage sur 5+6+5= **16 bits** permettant 2^{16}= **65536** couleurs différentes.

dnoir.py 001/011

```
from casioplot import *
L, H = 384, 192
def dnoir(c):
 for x in range(L):
  cd = [
   c[k]*x//(L-1)
   for k in range(3)
  ]
  for y in range(H):
   set_pixel(x,y,cd)
 show_screen()
```

FILE RUN SYMBOL CHAR A⇔a ▷





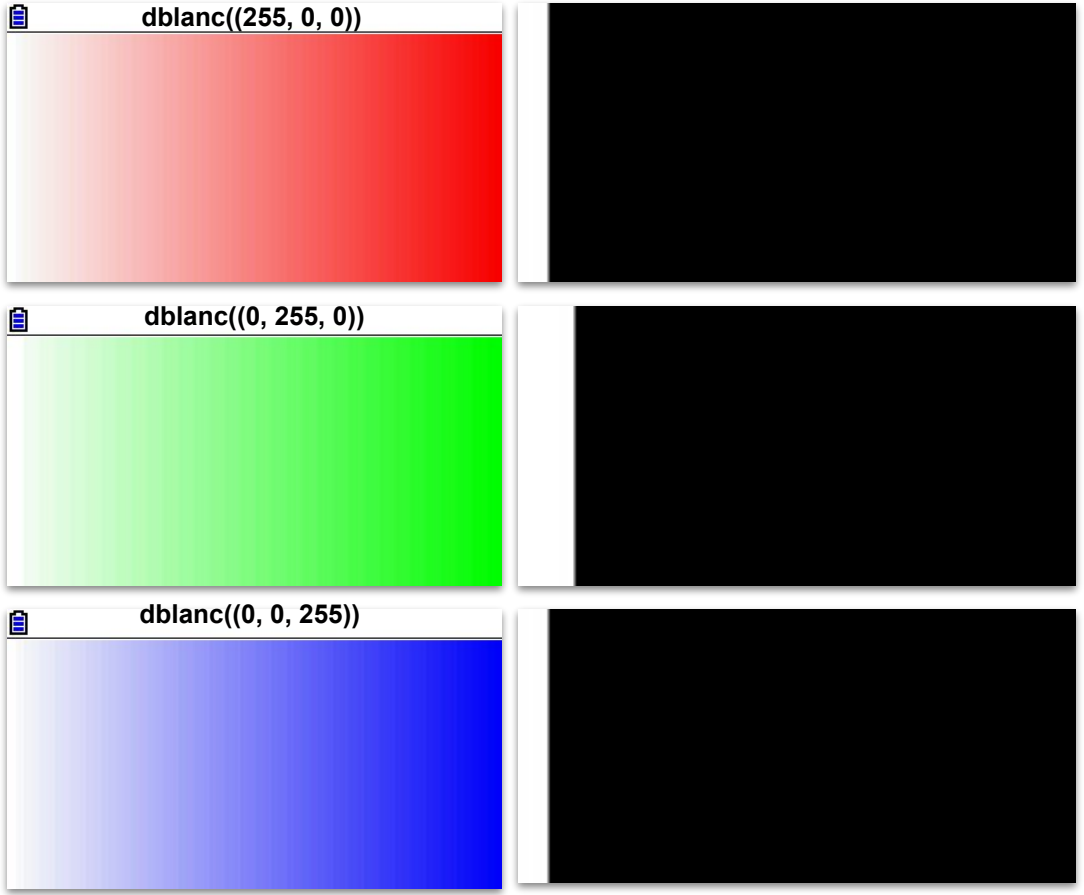
Les couleurs sur Graph 35+E II

L'écran **Graph 35+E II** affiche sur **1 bit** permettant 2^1= **2** couleurs différentes, noir ou blanc, selon la luminosité de la couleur demandée.

dblanc.py 001/011

```
from casioplot import *
L, H = 384, 192
def dblanc(c):
 for x in range(L):
  cd = [
   255-(255-c[k])*x//(L-1)
   for k in range(3)
  ]
  for y in range(H):
   set_pixel(x,y,cd)
 show_screen()
```

FILE RUN SYMBOL CHAR A⇔a ▷



Extension de casioplot avec les rectangles pleins

Étendons la bibliothèque casioplot avec une fonction **draw_rect()** permettant directement dessiner un rectangle de côtés parallèles aux bords de l'écran.

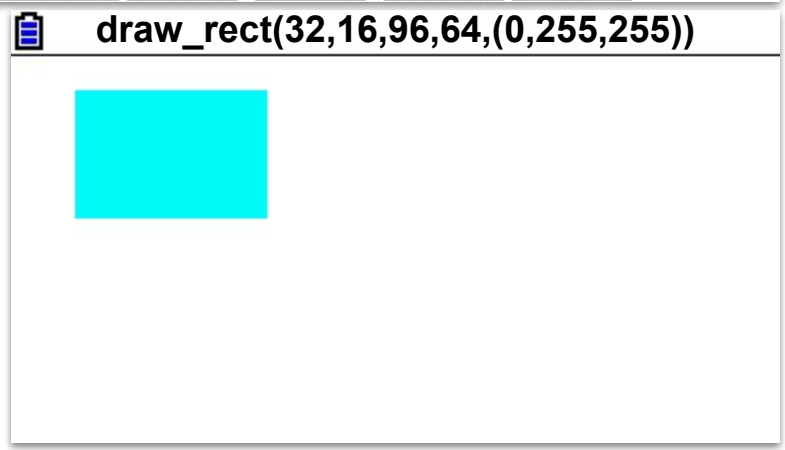
La fonction draw_rect() prend en paramètres :

- **(x, y)** : coordonnées du coin supérieur gauche du rectangle
- **l** : largeur du rectangle
- **h** : hauteur du rectangle
- **c** : la couleur de remplissage

drawrect.py 001/007

```
from casioplot import *
def draw_rect(x,y,l,h,c):
 for dy in range(h):
  for dx in range(l):
   tx = x + dx
   ty = y + dy
   set_pixel(tx,ty,c)
```

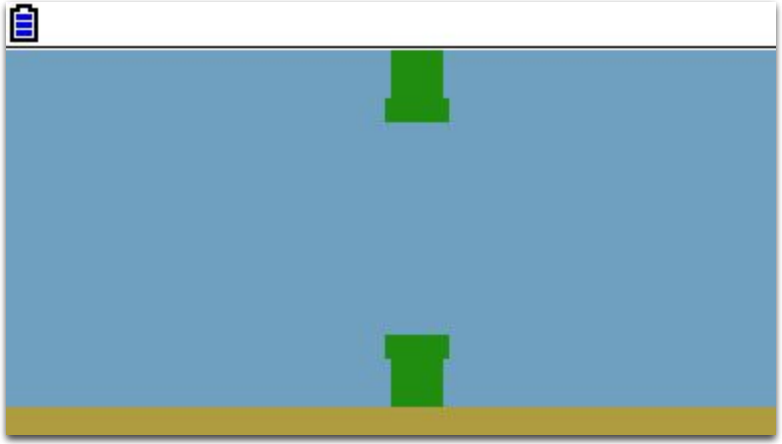
FILE RUN SYMBOL CHAR A⇔a ▷



Plantons le décor avec des rectangles pleins

La plupart des éléments graphiques de Flappy Bird peuvent être dessinés avec des rectangles pleins :

- fond d'écran / ciel
- barre d'état / sol
- corps des tuyaux
- bouches des tuyaux



decor1.py 001/016

```
from drawrect import *
L, H = 384, 192
draw_rect(0,0,L,H,(112,160,192))
HSOL = 14
draw_rect(0,H-HSOL,L,HSOL,(177,157,71))
LTUYAU,HTUYAU = 26,24
CTUYAU = (36,140,16)
x = L // 2
draw_rect(x,0,LTUYAU,HTUYAU,CTUYAU)
y2 = H-HSOL-HTUYAU
draw_rect(x,y2,LTUYAU,HTUYAU,CTUYAU)
LBOUCH,HBOUCH = 32,12
dx=(LBOUCH-LTUYAU)//2
draw_rect(x-dx,HTUYAU,LBOUCH,HBOUCH,CTUYAU)
draw_rect(x-dx,y2-HBOUCH,LBOUCH,HBOUCH,CTUYAU)
show_screen()
```

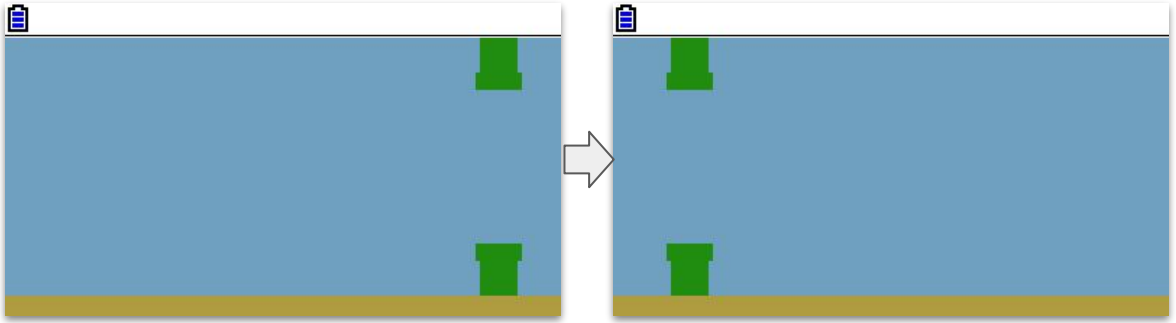
FILE RUN SYMBOL CHAR A⇔a ▷

Défilement horizontal du décor vers la gauche

Pour faire défiler horizontalement le décor, on peut mettre les affichages précédents dans une boucle faisant varier la valeur de la variable **x** :

- **diminuer** x pour un défilement vers la gauche
- **augmenter** x pour un défilement vers la droite

```
decor2.py      001/016
from drawrect import *
L, H = 384, 192
HSOL = 14
LTUYAU,HTUYAU = 26,24
CTUYAU = (36,140,16)
y2 = H-HSOL-HTUYAU
LBOUCH,HBOUCH = 32,12
dx=(LBOUCH-LTUYAU)//2
for x in range(L+dx,-LBOUCH,-1):
  draw_rect(0,0,L,H,(112,160,192))
  draw_rect(0,H-HSOL,L,HSOL,(177,157,71))
  draw_rect(x,0,LTUYAU,HTUYAU,CTUYAU)
  draw_rect(x,y2,LTUYAU,HTUYAU,CTUYAU)
  draw_rect(x-dx,HTUYAU,LBOUCH,HBOUCH,CTUYAU)
  draw_rect(x-dx,y2-HBOUCH,LBOUCH,HBOUCH,CTUYAU)
 show_screen()
FILE RUN SYMBOL CHAR A⇔a ▷
```



Défilement horizontal vers la gauche optimisé

À chaque étape, seule une minorité de pixels changent de valeur :

- la colonne de pixels du bord droit d'un rectangle passe à la couleur du fond
- la colonne de pixels juste à gauche d'un rectangle passe à la couleur du tuyau

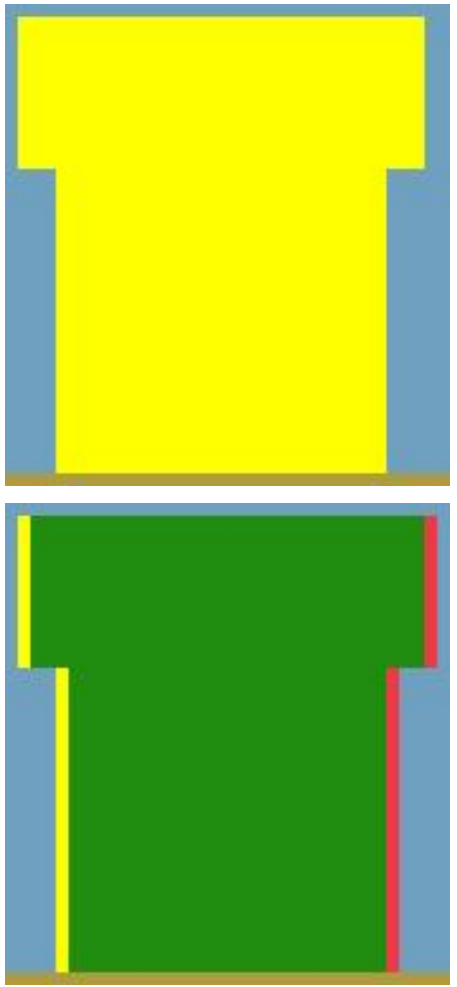
Après un dessin initial de l'ensemble des éléments, il suffit à chaque étape de redessiner ces seuls pixels :

2*2*(12+24)= 144 pixels au lieu de 73728 pixels soit 512 fois moins de données à traiter.

decor3.py 001/021

```
from drawrect import *
L, H = 384, 192
CFOND = (112,160,192)
draw_rect(0,0,L,H,CFOND)
HSOL = 14
draw_rect(0,H-HSOL,L,HSOL,(177,157,71))
LTUYAU,HTUYAU = 26,24
CTUYAU = (36,140,16)
y2 = H-HSOL-HTUYAU
LBOUCH,HBOUCH = 32,12
dx=(LBOUCH-LTUYAU)//2
for x in range(L, 0, -1):
 draw_rect(x+LTUYAU-1,0,1,HTUYAU,CFOND)
 draw_rect(x+LTUYAU-1,y2,1,HTUYAU,CFOND)
 draw_rect(x-dx+LBOUCH-1,HTUYAU,1,HBOUCH,CFOND)
 draw_rect(x-dx+LBOUCH-1,y2-HBOUCH,1,HBOUCH,CFOND)
 draw_rect(x,0,1,HTUYAU,CTUYAU)
 draw_rect(x,y2,1,HTUYAU,CTUYAU)
 draw_rect(x-dx,HTUYAU,1,HBOUCH,CTUYAU)
 draw_rect(x-dx,y2-HBOUCH,1,HBOUCH,CTUYAU)
 show_screen()
```

FILE RUN SYMBOL CHAR A⇔a ▷



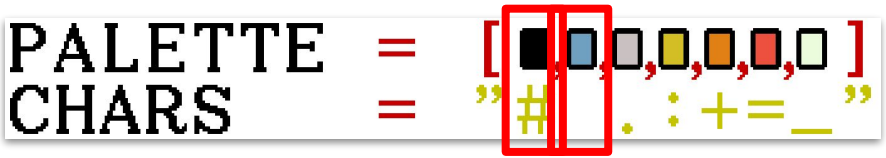
Dessin de Flappy Bird 

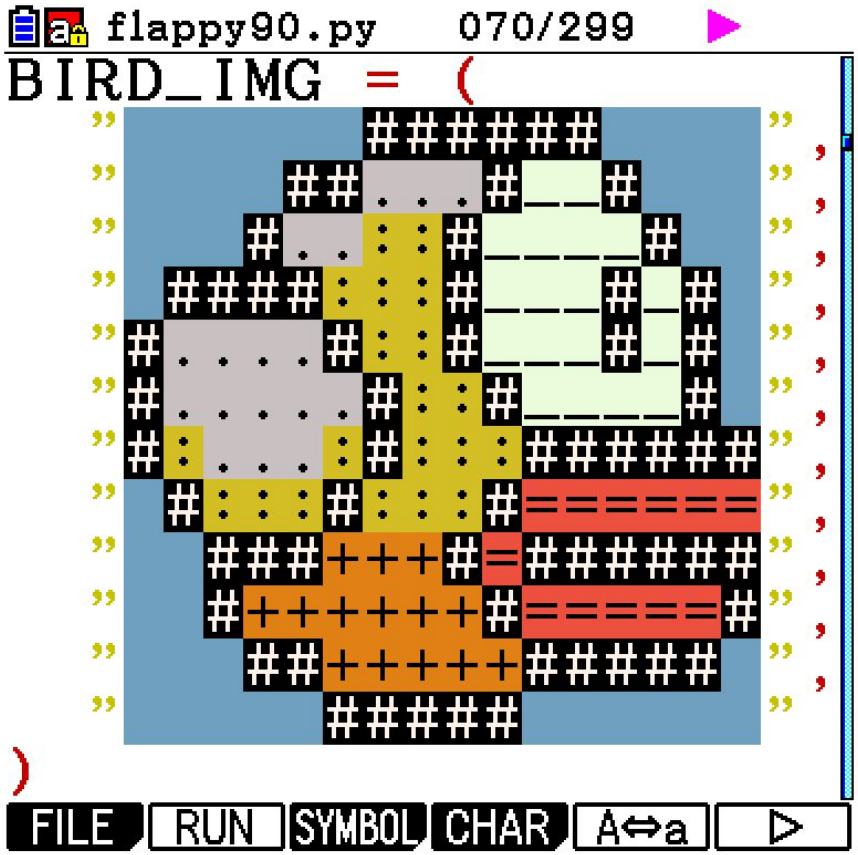
Une liste 2D de couleurs forme une image.

Mais une liste, de listes, de couleurs (listes RVB) ce serait illisible !

On fait donc une chaîne de caractères par ligne, et ensuite on associe une couleur à chaque caractère.

L'affichage est exactement comme `draw_rect()`, excepté que la couleur est prise dans `BIRD_IMG` au lieu d'être uniforme.

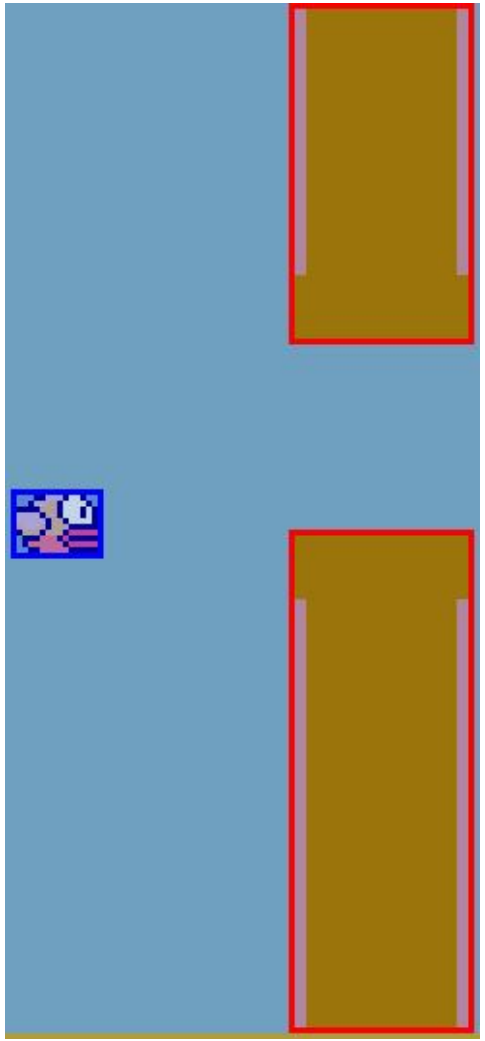




Collision entre oiseau et tuyaux

Pour simplifier le moteur physique, l’oiseau et les tuyaux peuvent être approximés par des rectangles.

Détecter une collision revient à se demander si il y a intersection non vide entre le rectangle de l’oiseau et un rectangle de tuyau.



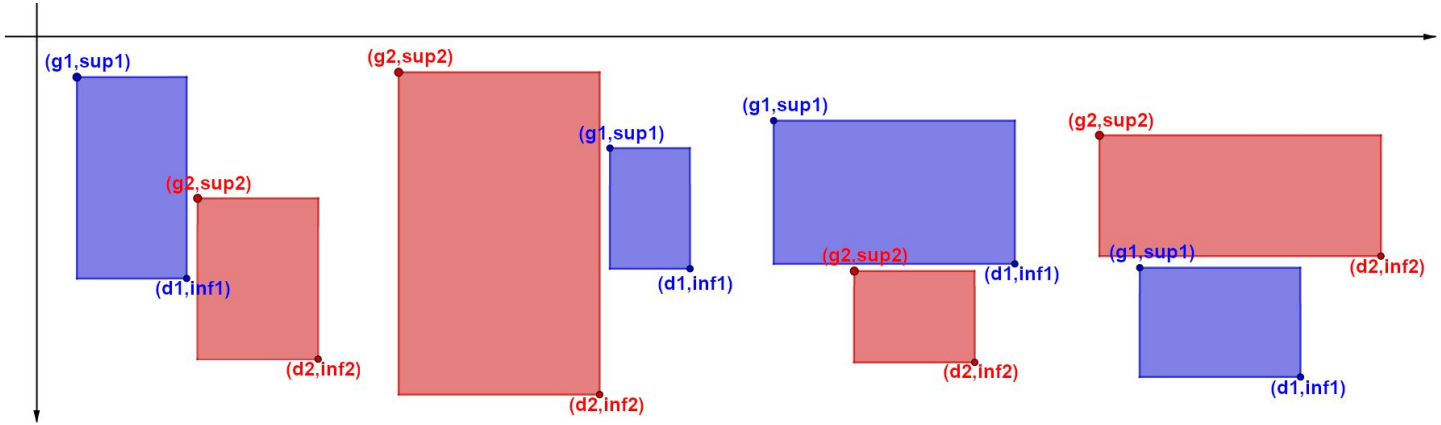
Non-intersection entre rectangles définis par 2 coins

Soit 2 rectangles :

- de coins supérieur gauche **(g1,sup1)** et inférieur droit **(d1,inf1)**
- de coins supérieur gauche **(g2,sup2)** et inférieur droit **(d2,inf2)**

Leur intersection est vide si au moins une des conditions suivantes est remplie :

- **d1<g2**
- **d2<g1**
- **inf1<sup2**
- **inf2<sup1**



Un test de non-intersection peut s'écrire : **d1<g2 or d2<g1 or inf1<sup2 or inf2<sup1**

Un test d'intersection s'écrit alors : **not(d1<g2 or d2<g1 or inf1<sup2 or inf2<sup1)**

ce qui sans négation se simplifie en : **d1>=g2 and d2>=g1 and inf1>=sup2 and inf2>=sup1**

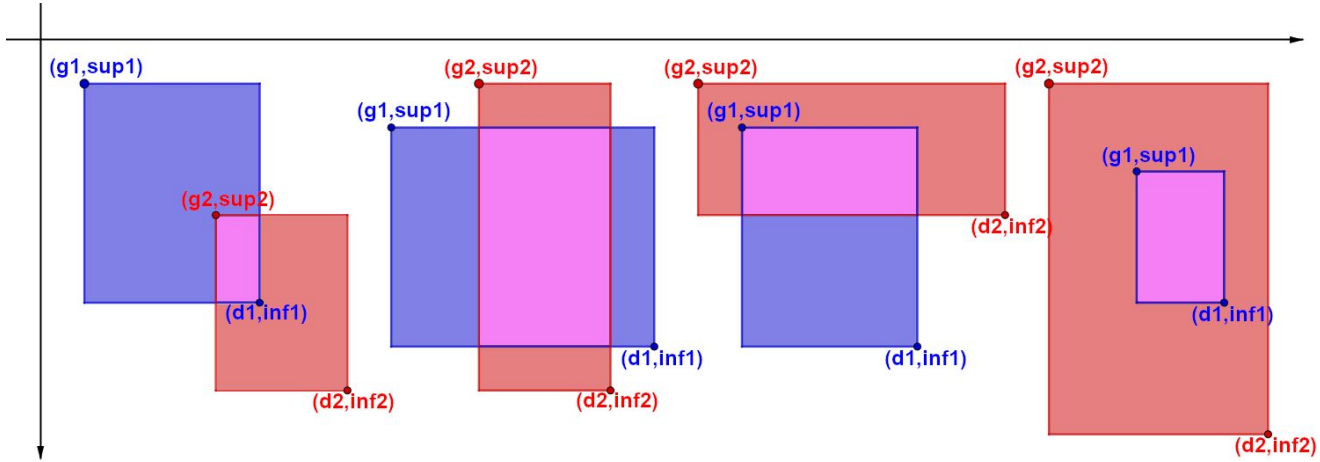
Intersection entre rectangles définis par 2 coins

Reprenons nos 2 rectangles :

- de coins supérieur gauche **(g1,sup1)** et inférieur droit **(d1,inf1)**
- de coins supérieur gauche **(g2,sup2)** et inférieur droit **(d2,inf2)**

Leur intersection si non vide est le rectangle de coins supérieur gauche **(g3,sup3)** et inférieur droit **(d3,inf3)**, avec :

- **g3=max(g1,g2)**
- **d3=min(d1,d2)**
- **sup3=max(sup1,sup2)**
- **inf3=min(inf1,inf2)**



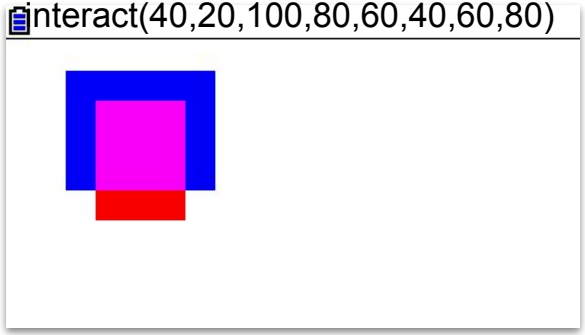
Un test peut alors consister à vérifier que ce rectangle existe :

g3 < d3 and sup3 < inf3

Intersection entre rectangles définis par 1 coin + 2 longueurs

Retranscrivons les 2 rectangles et donc le test avec les grandeurs utilisées jusqu'à présent par nos fonctions :

- de coin supérieur gauche **(x1,y1)**, largeur **l1** et hauteur **h1**
- de coin supérieur gauche **(x2,y2)**, largeur **l2** et hauteur **h2**



interect.py 001/012

```
from drawrect import *
def interect(x1,y1,l1,h1,x2,y2,l2,h2):
 clear_screen()
 maxgauch=max(x1,x2)
 mindroit=min(x1+l1,x2+l2)
 maxsup=max(y1,y2)
 mininf=min(y1+h1,y2+h2)
 draw_rect(x1,y1,l1,h1,(0,0,255))
 draw_rect(x2,y2,l2,h2,(255,0,0))
 draw_rect(maxgauch,maxsup,mindroit-maxgauch,mininf-maxsup,(255,0,255))
 show_screen()
 return maxgauch<mindroit and maxsup<mininf
```

FILE RUN SYMBOL CHAR A⇔a ▷

Du mouvement plan à la chute libre

2 grandeurs physiques pour un **mouvement** :

- la **vitesse** fait évoluer la position au cours du temps
- l'**accélération** fait évoluer la vitesse au cours du temps

6 variables pour contrôler le **mouvement plan** d'un point :

- (x, y) : position du point
- (vx, vy) : vitesses horizontale et verticale
- (ax, ay) : accélérations horizontale et verticale

```
mvplan.py      001/010
from casioplot import *
def mv(x,y,vx,vy,ax,ay):
    c = (0, 0, 255)
    for k in range(99):
        set_pixel(x,y,c)
        x += vx
        y += vy
        vx += ax
        vy += ay
        show_screen()
FILE RUN SYMBOL CHAR A⇔a ▷
```

Un point en **chute libre** n'est soumis qu'à une seule force, la gravité.

Dans l'atmosphère, si on peut négliger les frottements de l'air c'est une chute libre : il n'y a que la force poids, vertical et orienté vers le bas.
L'accélération est alors verticale (ax=0) et orientée vers le bas (ay>0).

Sans vitesse initiale (vx=0, vy=0) :
mv(32,0,0,0,0,1)

Avec vitesse initiale horizontale vers la droite (vx>0, vy=0) :
mv(32,0,5,0,0,1)

Merci pour votre attention !

Téléchargement des scripts et ressources:

- https://www.casio-education.fr/

Animé par Planète Casio et TI-Planet :

- https://www.planet-casio.com
- https://tiplanet.org